Advanced practical **Programming** for Scientists

Thorsten Koch

Zuse Institute Berlin

TU Berlin

WS2014/15

http://0hh1.com/

```
set I := { 1 .. size };

var x[I][I] binary;

subto c1: # rows half / half
forall <i> in I do
   sum <j> in I: x[i][j] == size / 2;

subto c2: # cols half / half
forall <i> in I do
   sum <j> in I: x[j][i] == size / 2;

subto c1: # never three equal color in a row
forall <i> in I do
   forall <j> in I \ { 1, 2 } do
      1 <= x[i][j - 2] + x[i][j - 1] + x[i][j] <= 2;

subto c1: # never three equal color in a col
forall <i> in I do
   forall <j> in I \ { 1, 2 } do
      1 <= x[j - 2][i] + x[j - 1][i] + x[j][i] <= 2;
```

n & -n

How does it work?

0001 & 1110 + 1 = 1111 = 0001

0010 & 1101 + 1 = 1110 = 0010

0011 & 1100 + 1 = 1101 = 0001

0100 & 1011 + 1 = 1100 = 0100

0101 & 1010 + 1 = 1011 = 0001

…

How does this work?

```
if (updatemask & 0xffff0000)
   colidx += 16;
if (updatemask & 0xff00ff00)
   colidx += 8;
if (updatemask & 0xf0f0f0f0)
   colidx += 4;
if (updatemask & 0xcccccccc)
   colidx += 2;
if (updatemask & 0xaaaaaaaa)
   colidx += 1;
```

Exercise

```
if (updatemask & 11111111111111110000000000000000b)
   colidx += 16;
if (updatemask & 11111111000000001111111100000000b)
   colidx += 8;
if (updatemask & 11110000111100001111000011110000b)
   colidx += 4;
if (updatemask & 11001100110011001100110011001100b)
   colidx += 2;
if (updatemask & 10101010101010101010101010101010b)
   colidx += 1;
```

Black-Box-Testing

is a method of software testing that is applied without any knowledge of the inner working of the system to be tested.

It is limited to functionality related tests, i.e. the test cases are derived from the requirements not from the implementation of the test object.

The program is treated as a black box. Only properties that are visible to the outside are tested.

Example: Running test inputs into a system/function and checking whether the outcome is as expected.

Does it do the right thing?

Disadvantage: *Completeness impossible*

White-Box-Testing

(also known as **clear box testing**, **glass box testing**, **transparent box testing**, and **structural testing**) tests the internal structures or workings, as opposed to its functionality. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

Examples:

- *Line coverage: Execution of all source lines*
- *Instruction (node) coverage: Execution of all instructions*
- *Branch (edge) coverage: All branches of the control flow are executed*
- *Path coverage: Execution of all paths through a module*

Does it work in the intended way?

Disadvantage: intended and correct are not the same thing.

Do the tests first.

How to do this in our case?

How to do this in more complicated cases?

How to do this in research,
where you are the first to answer a question?

How to grade you?

I think I have to measure simething like the amount of learning? / understanding? / knowledge? you possess after the course? / have gained from the course?

Measurement is defined as:

The process of assigning symbols, usually numbers,
to represent an attribute of the entity of interest, by rule.

Maybe I grade you by the, should we call it quality?, of the code you submit to the exercises.

How to we do this?

Software Metrics

A **software metric** is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development.

The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

What properties of a piece of software might have?
(that we can measure by rule)

Cyclomatic Complexity

- **Cyclomatic complexity** is a software metric (measurement).
- It was developed by Thomas J. McCabe, Sr. in 1976 and is used to indicate the complexity of a program.
- It is a quantitative measure of the complexity of programming instructions.
- It directly measures the number of linearly independent paths through a program's source code.
- Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.
- One testing strategy, called basis path testing by McCabe who first proposed it, is to test each linearly independent path through the program; in this case, the number of test cases will equal the cyclomatic complexity of the program.

Example

```
const char* weekdayname(int num)
{
  switch(num) {
    case 1: return "Montag";
    case 2: return "Dienstag";
    case 3: return "Mittwoch";
    case 4: return "Donnerstag";
    case 5: return "Freitag";
    case 6: return "Samstag";
    case 7: return "Sonntag";
  }
  return "(unbekannter Wochentag)";
}

CC = 8
```

```
const char* weekdayname(int num)
{
   const char* tage[] =
   { "Montag","Dienstag","Mittwoch",
     "Donnerstag","Freitag",
     "Samstag", "Sonntag“
   };
   int len = sizeof (tage);

   if ((num >= 1) && (num <= len))
      return tage[num - 1];

   return "(unbekannter Wochentag)";
}

CC = 3
```

What is the MIPLIB

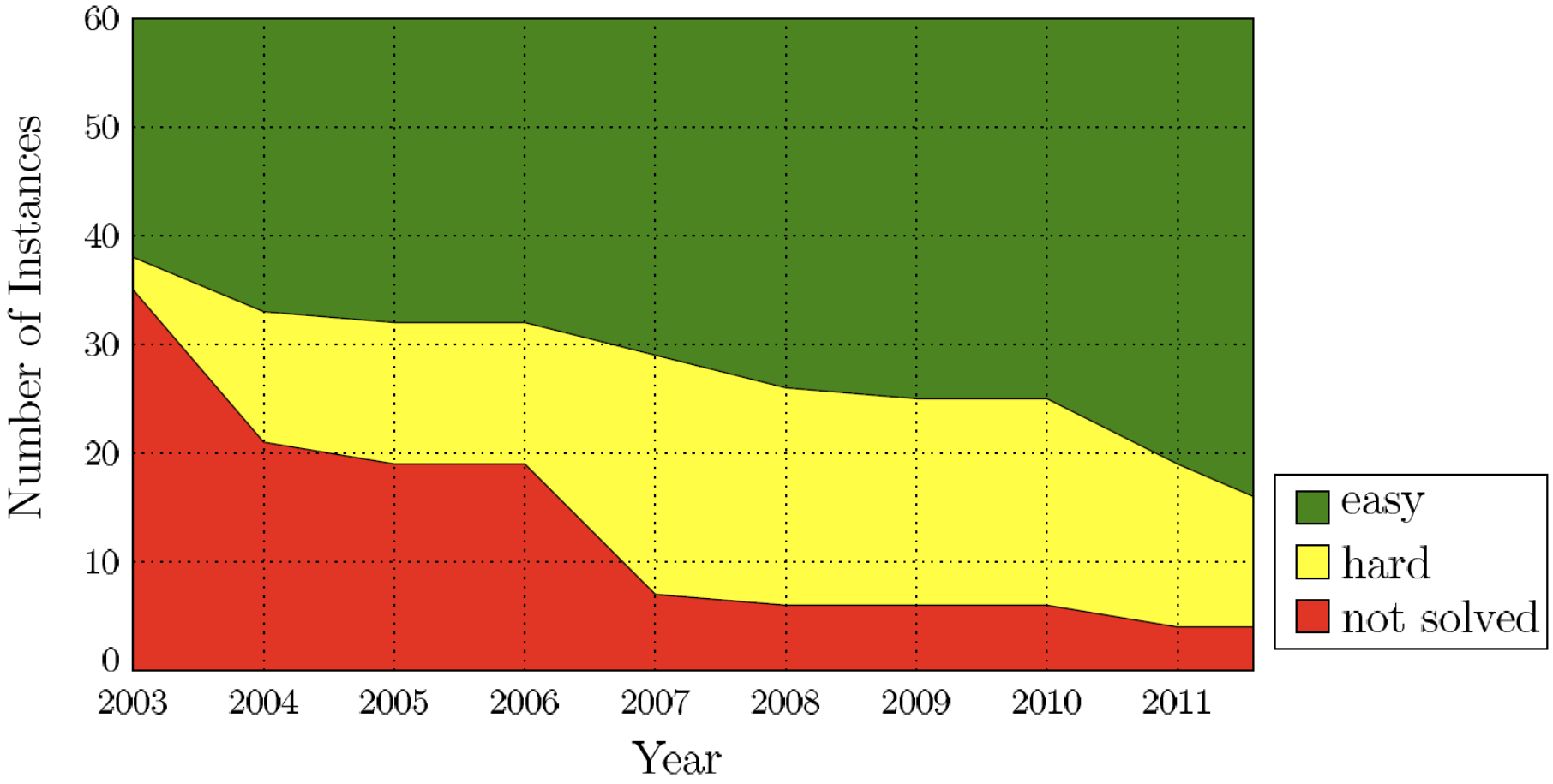
Version	Year	#	Who	Reference
1	1991	61	Bixby, Boyd, Indovina	
2	1992	61	Bixby, Boyd, Indovina	SIAM News 25, 15 (1992)
3	1996	65	Bixby, Ceria, McZeal, Savelsberg	Optima 58, 12-15 (1998)
4	2003	60	Achterberg, Koch, Martin	ORL 34(4) 361-372 (2006)
5	2010	87/361	Koch, Achterberg, Andersen, Bastert, Berthold, Bixby, Danna, Gamrath, Gleixner, Heinz, Lodi, Mittelmann, Ralphs, Salvagnin, Steffy, Wolter	MPC 3, 103-163 (2011)

The MIPLIB is a diverse collection of challenging real-world mixed integer programming (MIP) instances from various academic and industrial applications suited for benchmarking and testing of MIP solution algorithms.

The *Benchmark* set consist only of instances that could be solved in 2010 within two hours on a high-end personal computer by at least two MIP solvers.

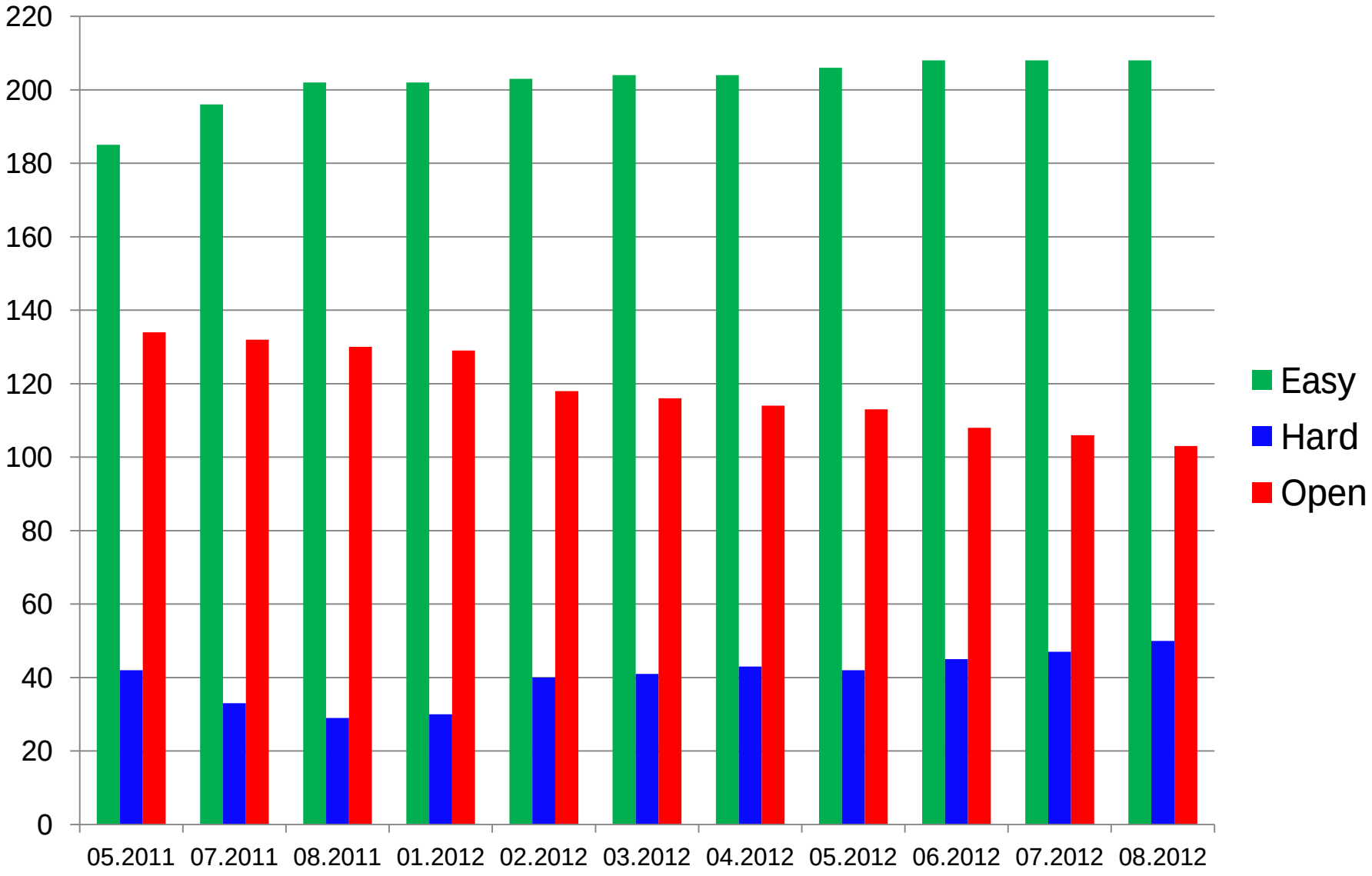
MIPLIB 5 still contains 2 instances from Version 2, 7 instances from Version 3, and 17 instances from Version 4.

Development of MIPLIB 2003



easy can be solved within an hour on a contemporary pc with a state-of-the-art solver
hard are solvable but take a longer time or require specialized algorithms
open, *i.e. not solved* instances for which the optimal solution is not known

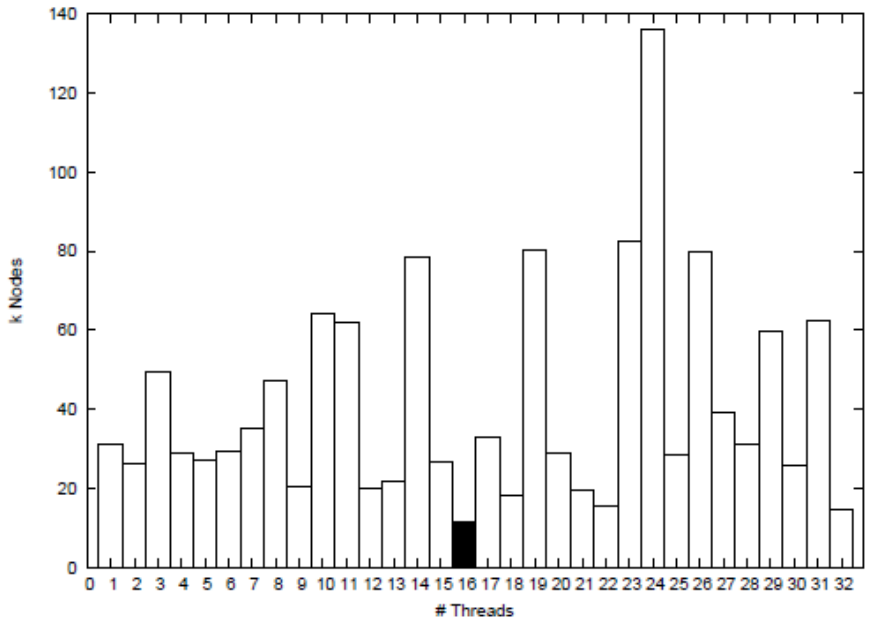
Development of MIPLIB 2010

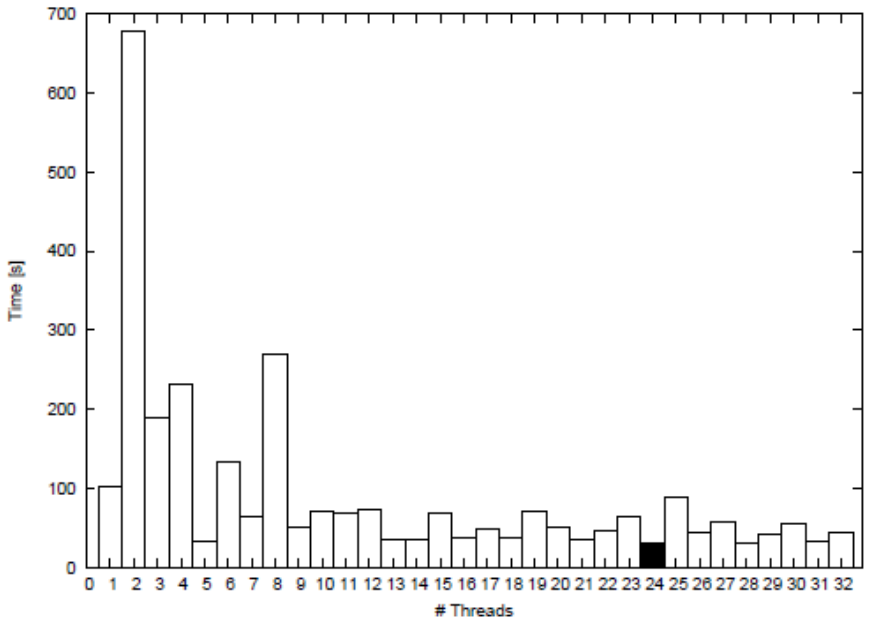


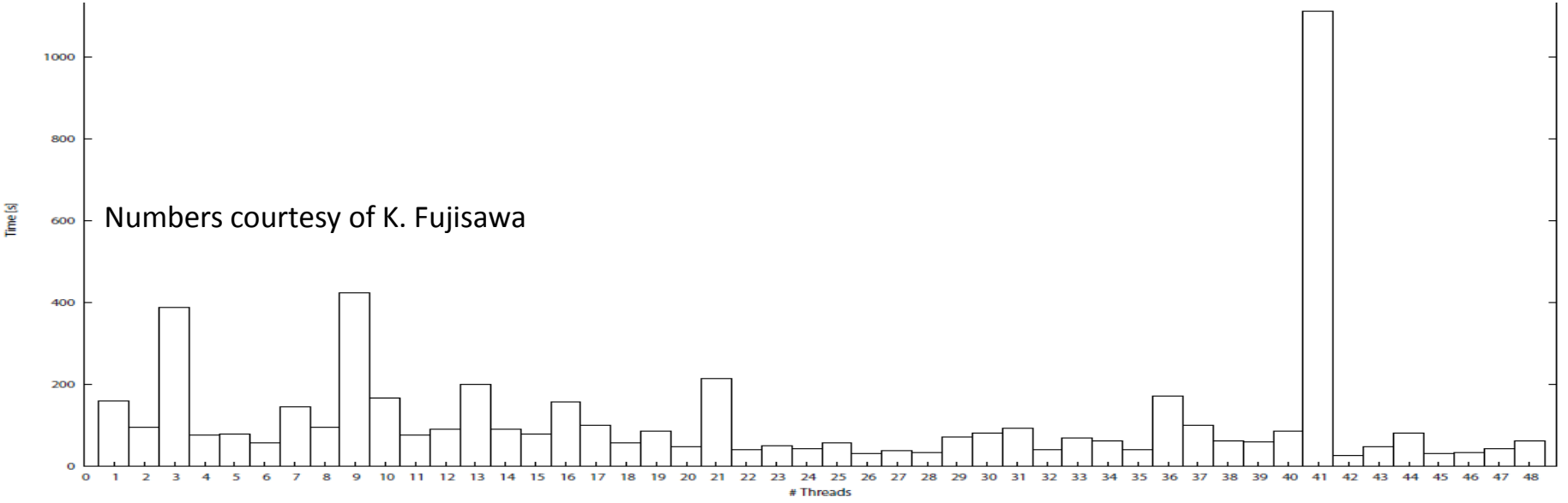
Some MIP Solvers then and now

Solver	Version 2011	Version 2012	URL
IBM CPLEX	12.2	12.4	www.cplex.com
Gurobi	4.5 beta0	5.0	www.gurobi.com
FICO XPress-MP	7.2 RC	7.2	www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx
SCIP/spx	2.0.1	3.0.0	scip.zib.de
CBC	2.6.4	2.7.7	projects.coin-or.org/Cbc
Further solvers			
Glpk		4.47	www.gnu.org/software/glpk/glpk.html
lp_solve		5.5	lpsolve.sourceforge.net
Lindo		7.1	www.lindo.com
Minto		3.1	coral.ie.lehigh.edu/~minto
Symphony		5.4	projects.coin-or.org/SYMPHONY

Performance relative to number of threads

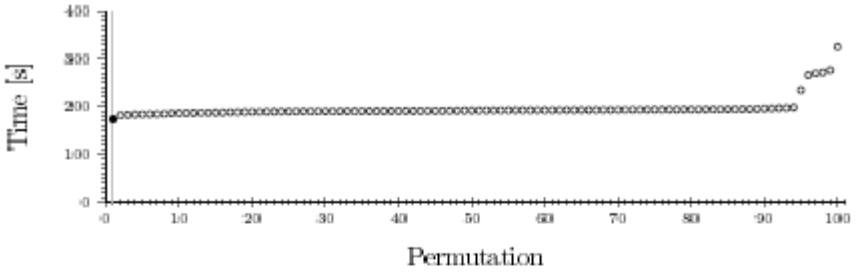




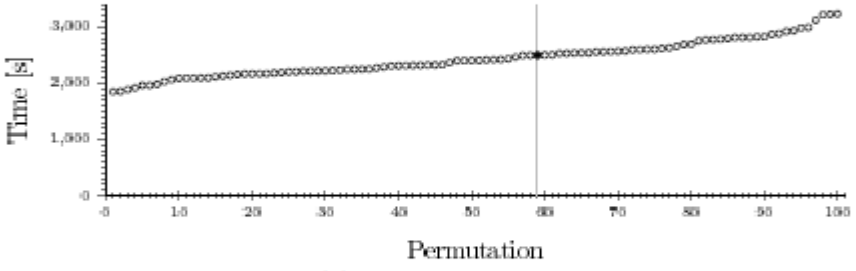


Numbers courtesy of K. Fujisawa

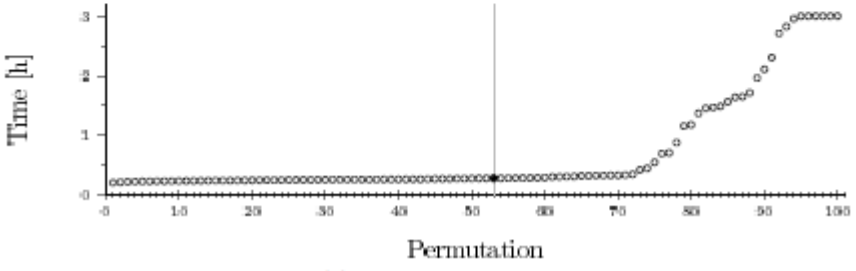
Performance variation due to permutation



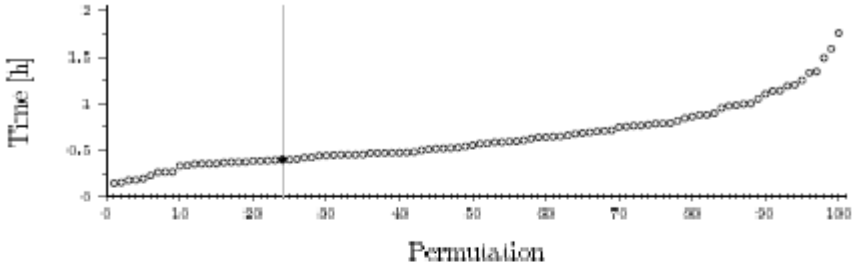
(a) Instance ex9



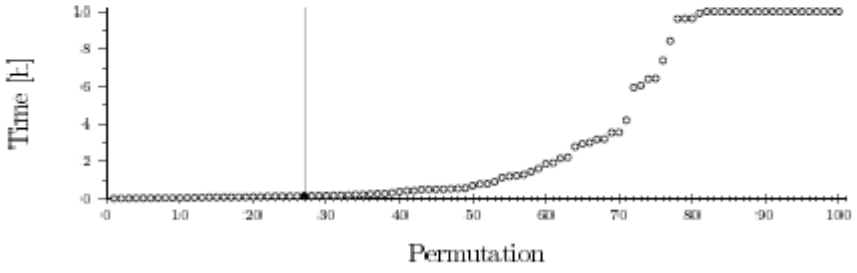
(b) Instance pg5_34



(c) Instance neos13



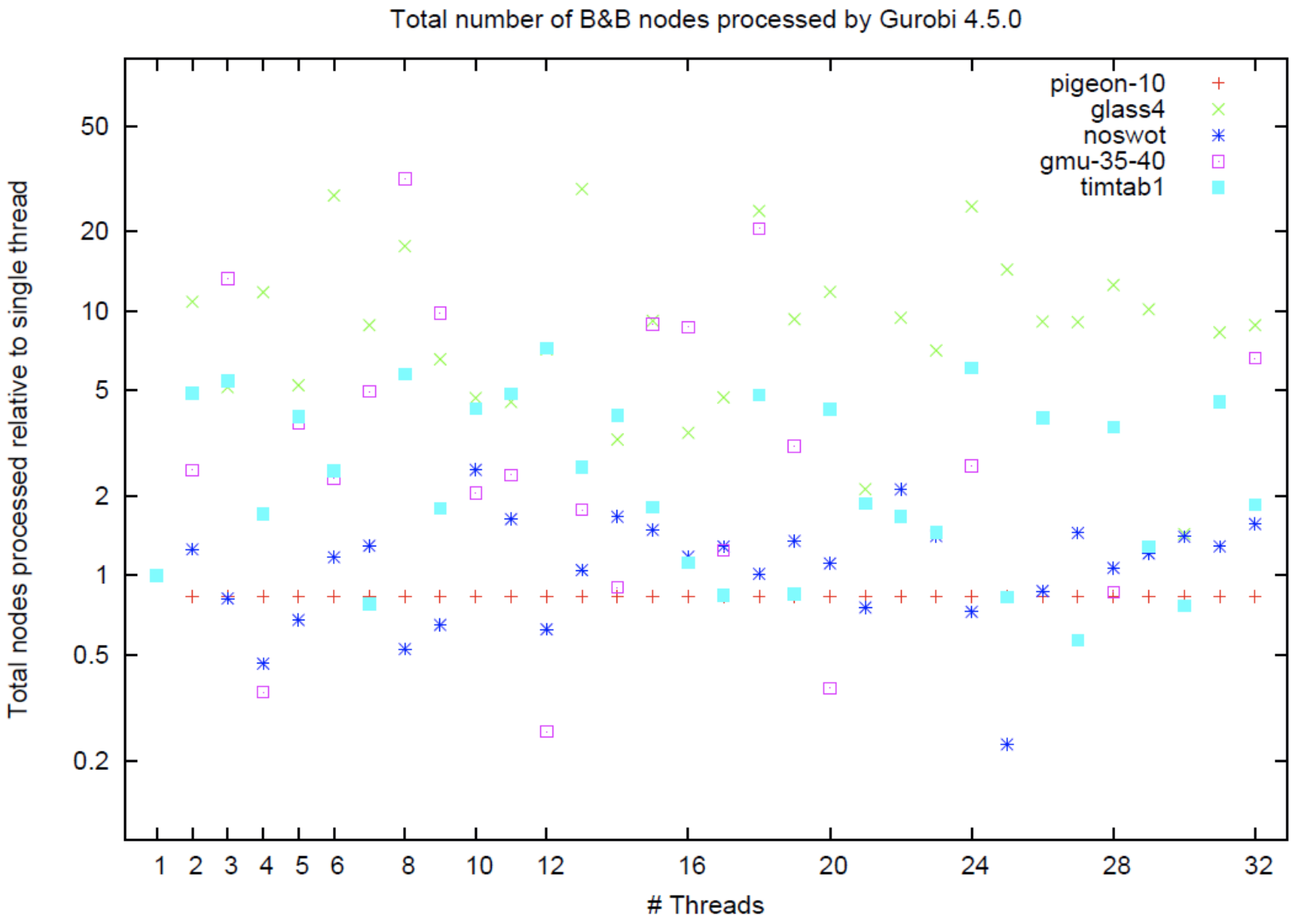
(d) Instance bnatt350



(e) Instance enlight13

Fig. 3: Solution times for 100 permutations

B&B nodes


Total number of B&B nodes processed by Gurobi 4.5.0
pigeon-10
glass4
noswot
gmu-35-40
timtab1
Total nodes processed relative to single thread
50
20
10
5
2
1
0.5
0.2
1
2
3
4
5
6
8
10
12
16
20
24
28
32
Threads

How to compare MIP solver performance?

There are two components to it:

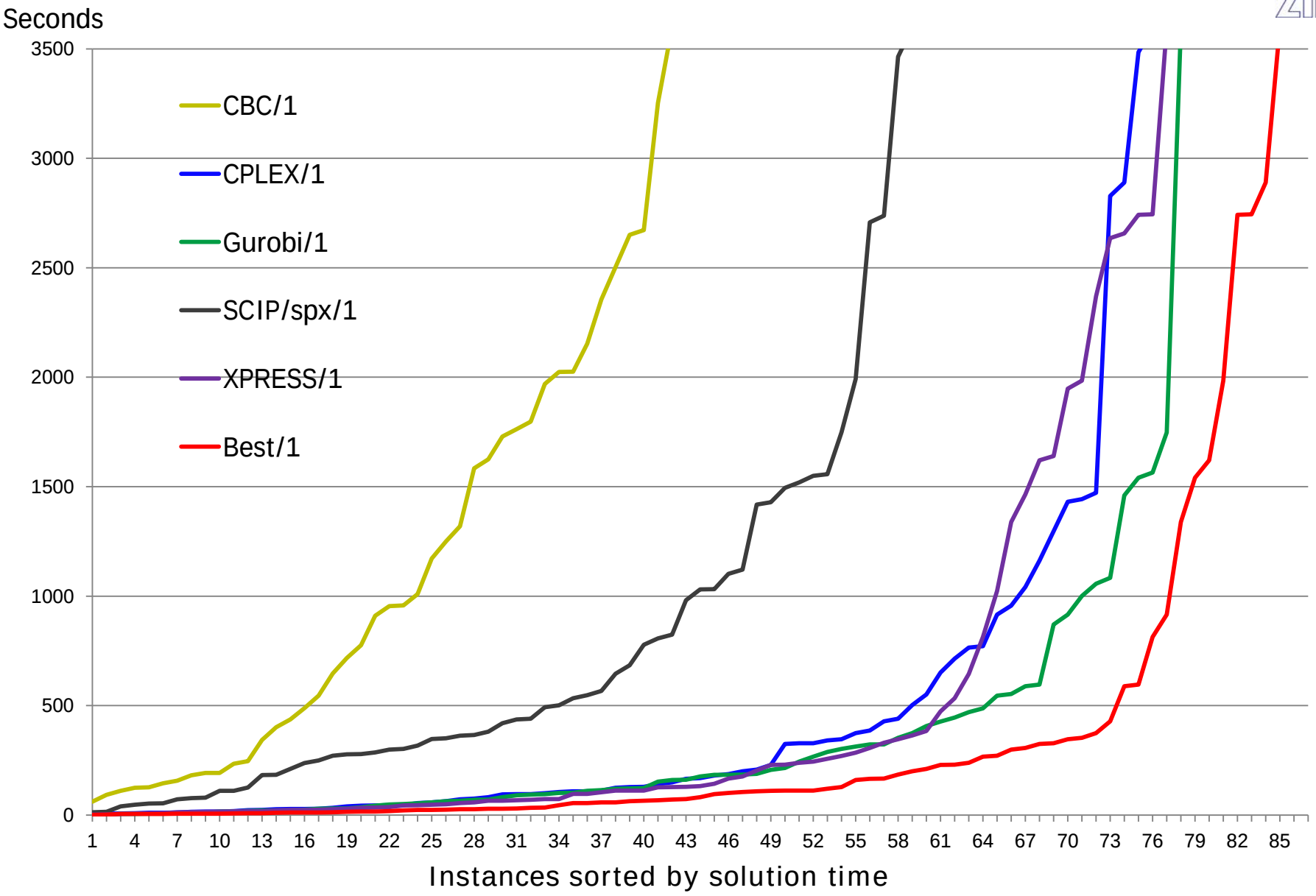
- Ability to solve an instance (within some timeframe)
- Time to solve (heuristics see also primal integral)

You have always to see both numbers!

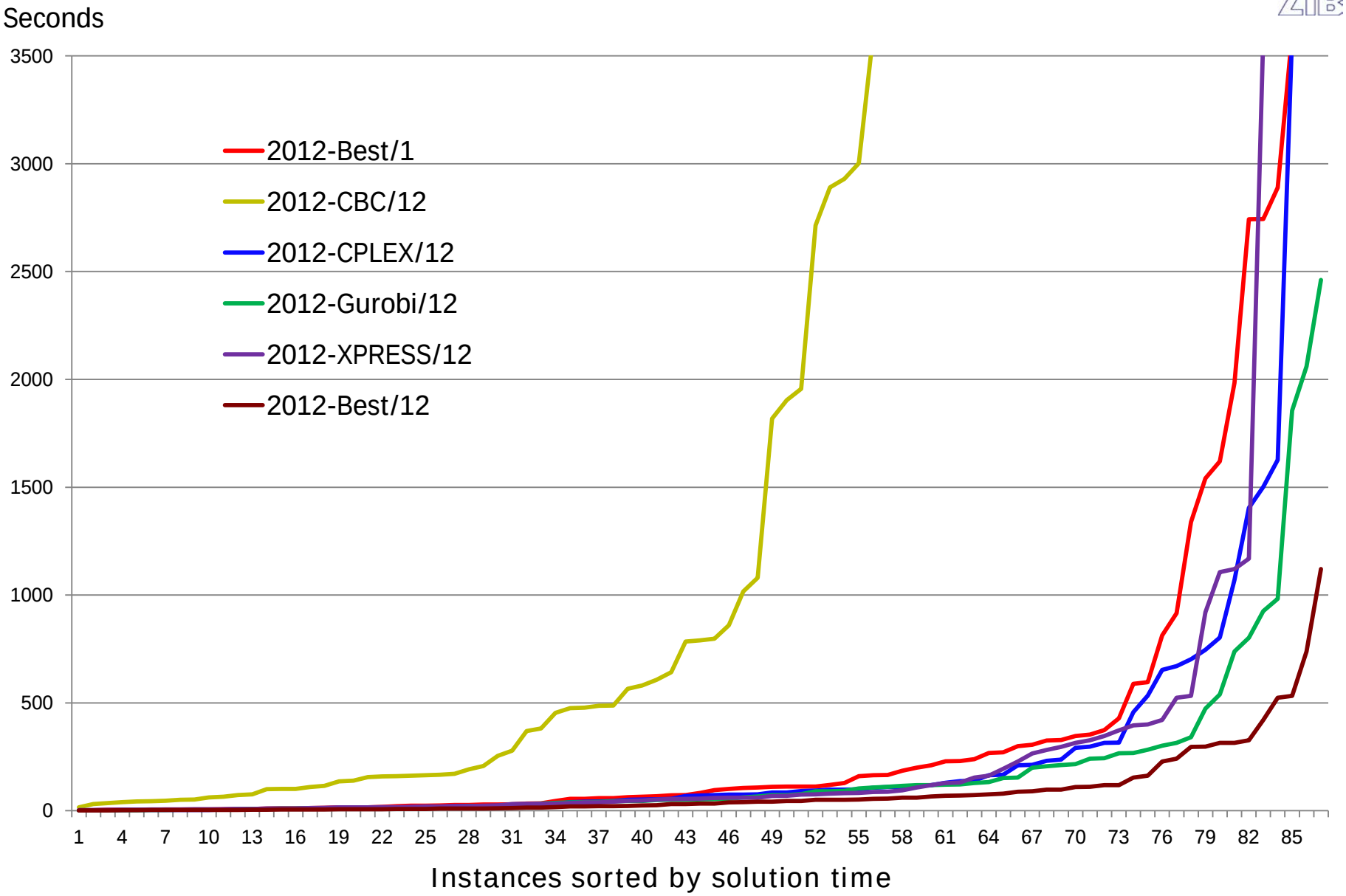
If two solvers are compared there are two possibilities:

1. Both can solve all instances.
2. There are instances only one solver can solve
3. There are instances both cannot solve

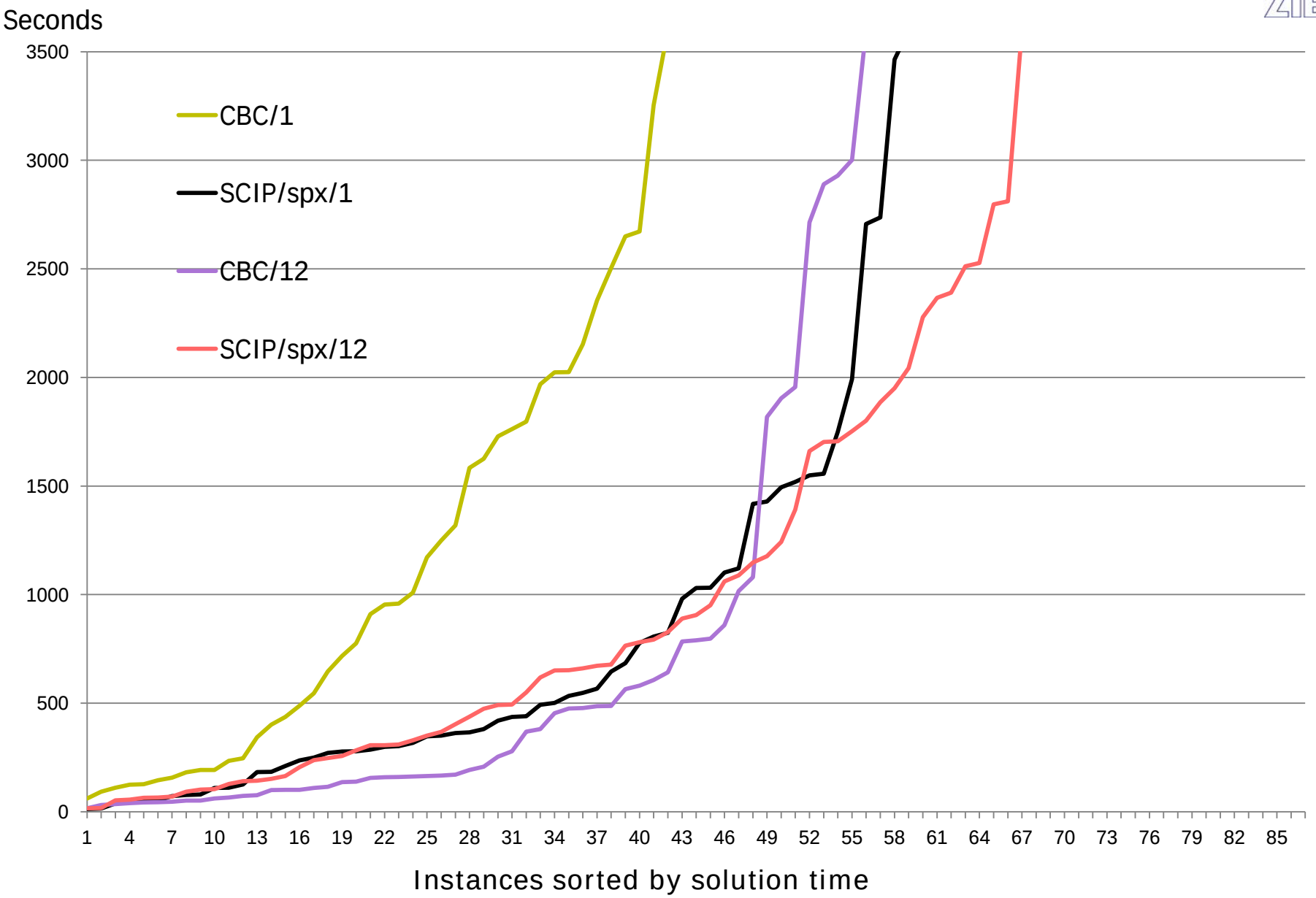
2012 Performance on MIPLIB 2010 / 1 thread



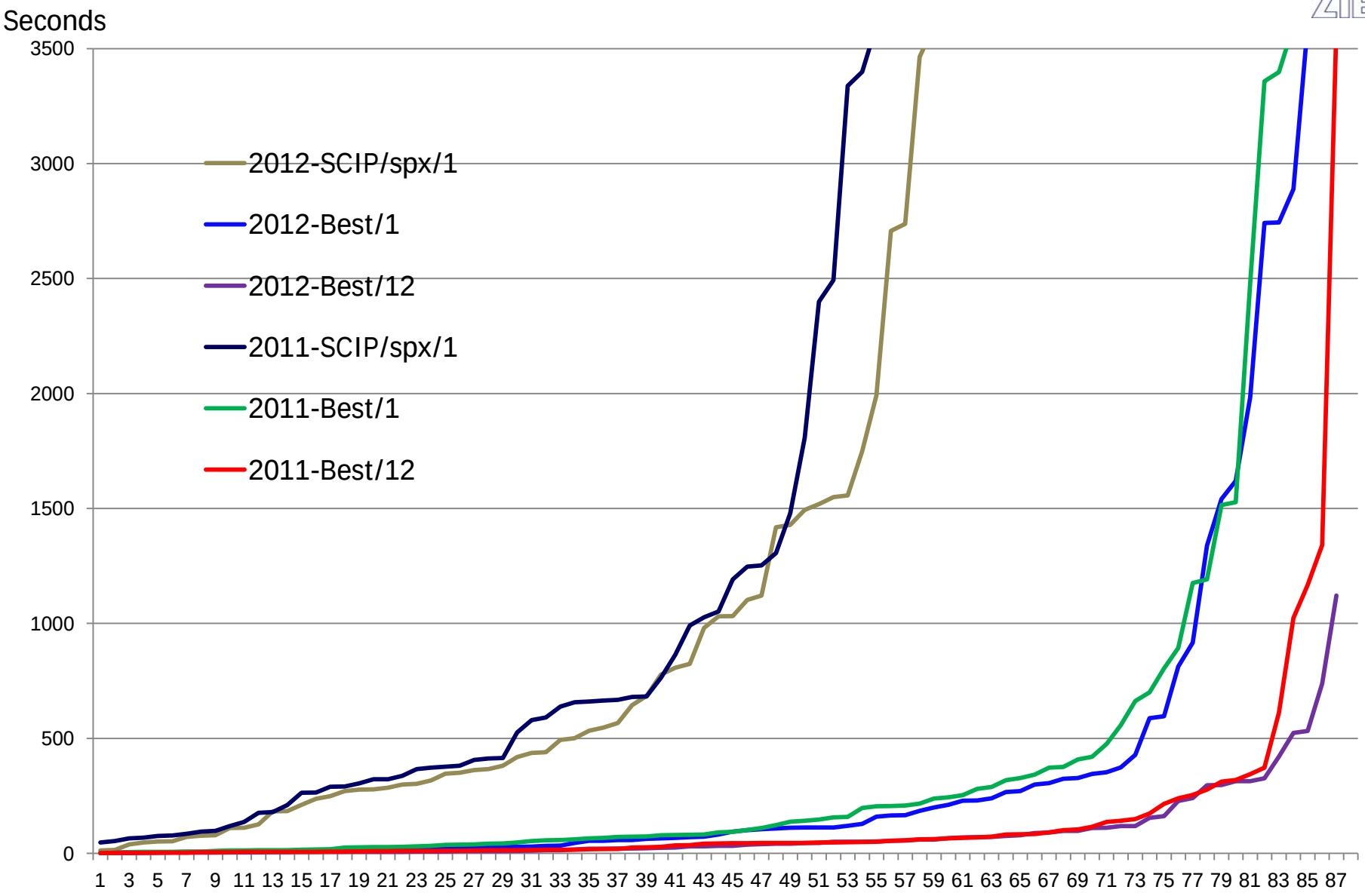
2012 Performance on MIPLIB 2010 / 12 threads



2012 Speedup 1:12 threads CBC/SCIP



MIPLIB Performance 2011:2012

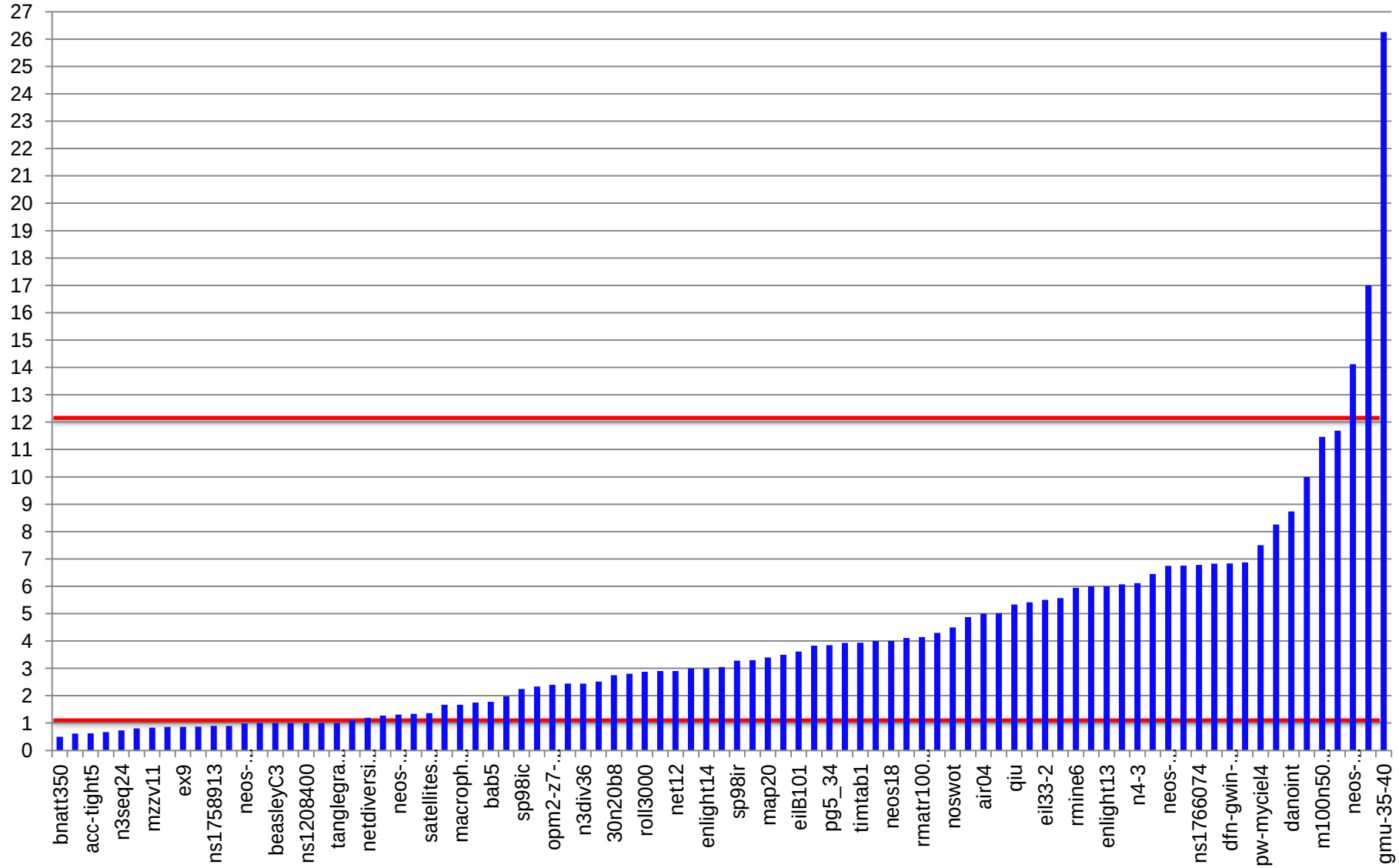


Progress in Numbers

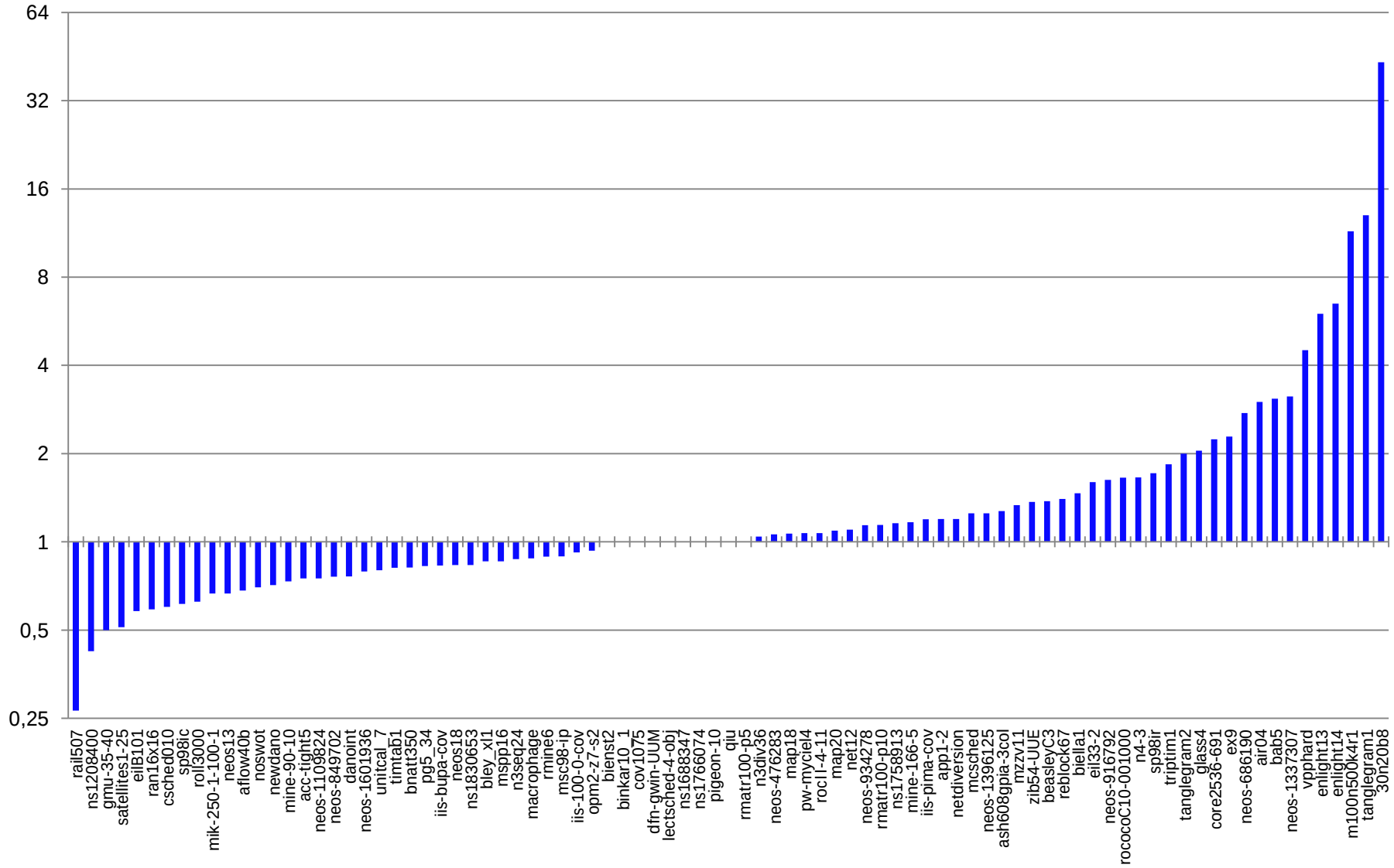
	2011	2012	Speed-up
Best/1 geom. mean time [s]	105	81	**23 %**
Best/12 geom. mean time [s]	34	28	**17 %**
Best/1 not optimal [#]	4	3	
Best/12 not optimal [#]	1	0	
Speedup 1 to 12 [×]	3.1	2.8	
Max(C,G,X)/Min(C,G,X) 1 thr.	600	259	
Max(C,G,X)/Min(C,G,X) 12 thr.	1138	107	
Best/1 geom. mean nodes [#]	2108	1958	8%
Best/12 geom. mean nodes [#]	2570	2406	7%
Max(C,G,X)/Min(C,G,X) 1 thr.	40,804	32,600	
Max(C,G,X)/Min(C,G,X) 12 thr.	10,499	887,849	

Single solvers have speed-ups up to 42%

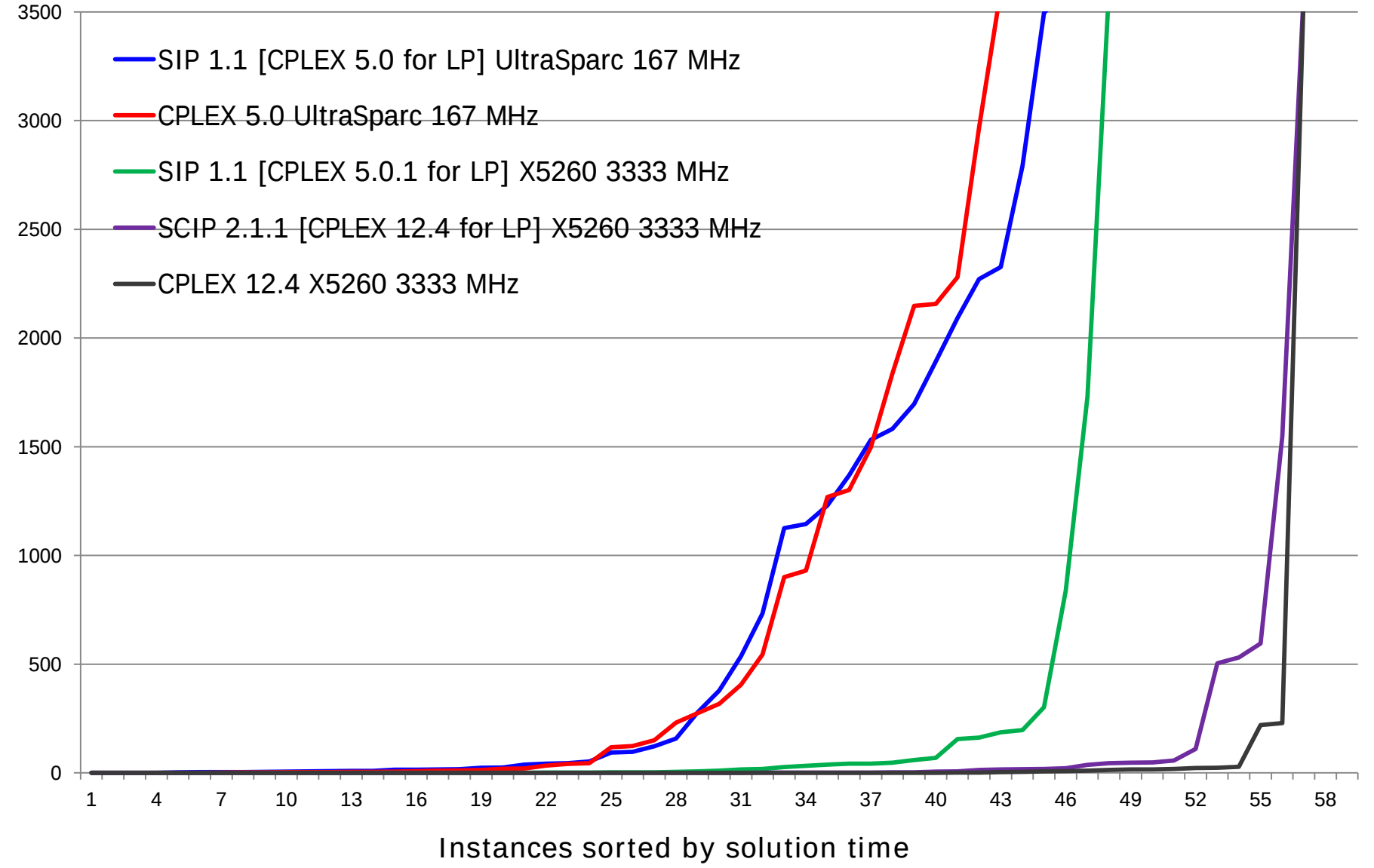
2012 Parallel speedup Best 1/12 per instance



Speedup Best/12 2011:2012 per instance



1998 Benchmark Set



Read **Ada for the C++ or Java Developer**